Apple Assembly Line

$1.80

Volume 5 -- Issue 5 February, 1985

65816 News -- Talked with Bill Mensch a few days ago, and he expects full production in just a few weeks. There should be a lot of sources soon. Bill has a few more great chips in mind, upgrading the 6502 family even further.

David Eyes is writing a detailed programmer's reference manual for the 65816, to be published about July by Brady. Bill says it should answer all our questions. I'll be reviewing it as soon as possible.

We hear of a 6MHz 65816 board with 256K RAM for plugging into Apples. Let you know when we learn more details.

Woz News -- We hear Steve, Wendell Sander (/// designer), and Joe Ennis (//c designer have teamed up to form a new enterprise, outside Apple, with plans to produce a device for the home video market.

Apple II Forever College -- If you would like in-depth training in Cupertino, $500 buys 3 days under the masters. One session starts March 6th, another May 8th. Call Marian Djurovich at (408) 973-6411 for details.

18-Digit Arithmetic, Part 10.....Bob Sander-Cederlof

At least one error crept into the PRINT USING program we printed last month. A line should be inserted to correct the problem:

```
3045     JSR PRUS.CLEAR    YES, NEW FIELD
```

This is what I expect to be the final installment of the DP18 series. Some of you have been typing in and trying out the various installments, and others buying the source code on the various quarterly disks. We plan to make the composite DP18 source available at a reasonable price: all parts will be properly integrated as a set of 12 source files, ready to assemble with the S-C Macro Assembler. The disk will also include example programs illustrating the various features, the object file of DP18, and a loader program for installing DP18. The price for all of it, on one diskette, will be $50.

Normal Applesoft INPUT statements can be written in several ways. An optional quotation can be used for a prompting message; if one is used a semicolon must follow the quotation. A list of one or more variables follows.

```
INPUT variable
INPUT "quote";variable
```

In DP18 we implemented the two forms of the INPUT statement shown above, except that only a single variable may be used in each statement. We also implemented two additional kinds of INPUT statements. INPUT# statements allow expressions to be entered during execution. INPUT$ statements allow picture-controlled input.

```
INPUT # variable
INPUT # "quote";variable
INPUT $ string,variable-list
```

The INPUT# statement allows you to read expressions and evaluate them during an INPUT operation. This can greatly simplify entering some numbers. For example, one-third can be entered as either ".33333333333333333333" or simply as "1/3". You can enter values such as SQR(2), 2*PI, and so on. You can even refer to variables used in the program. After you have entered the expression and typed RETURN, DP18 calls on Applesoft to tokenize the line, evaluates the expression to a numeric value, and stores the value in the INPUT variable your program specified.

We call the INPUT$ statement "INPUT using". It is analogous to "PRINT using", or the PRINT$ statement discussed last month. All characters in the INPUT$ picture are proccessed the same as for PRINT$ until characters defining a numeric or string field are encountered. Then the magic begins....

For a numeric field, underlines are printed to indicate digit positions. The cursor is placed after the last underline. If there is a decimal point in the picture it will be printed. A

S-C Macro Assembler Version 2.0.....$100
Version 2.0 Upgrade Kit (for 1.0/1.1/1.2 owners).....$20
Source Code for Version 1.1 (on two disk sides).....$100
Full Screen Editor for S-C Macro (with complete source code).....$49
S-C Cross Reference Utility for S-C Macro (without source code).....$20
Source Code for S-C Cross Reference Utility.....additional $50
DISASM Dis-Assembler (RAK-Ware).....$30
Source Code for DISASM.....additional $30
S-C Word Processor (with complete source code).....$50
DP18 Source and Object.....$50
Double Precision Floating Point for Applesoft (with source code).....$50
S-C Documentor (complete commented source code of Applesoft ROMs).....$50
Source Code of //e CX & F8 ROMs on disk.....$15

(All source code is formatted for S-C Macro Assembler. Other assemblers require some effort to convert file type and edit directives.)

AAL Quarterly Disks.....each $15, or any four for $45

		Jan-Mar	Apr-Jun	Jul-Sep	Oct-Dec
Each disk contains	1980	-	-	-	1
the source code from	1981	2	3	4	5
three issues of AAL,	1982	6	7	8	9
saving you lots of	1983	10	11	12	13
typing and testing.	1984	14	15	16	17

AWIIe Toolkit (Don Lancaster, Synergetics).....$39
Visible Computer: 6502 (Software Masters).....(reg. $50) $45
ES-CAPE: Extended S-C Applesoft Program Editor (new price, was $60)..$40
"Bag of Tricks", Worth & Lechner, with diskette.....($39.95) $36

Blank Diskettes (Verbatim)..... package of 20 for $32
(Premium quality, single-sided, double density, with hub rings)
Vinyl disk pages, 6"x8.5", hold two disks each.....10 for $6
Diskette Mailing Protectors (hold 1 or 2 disks).....40 cents each
or $25 per 100
(These are cardboard folders designed to fit into 6"X9" Envelopes.)
Envelopes for Diskette Mailers..... 6 cents each
quikLoader EPROM System (SCRG).....($179) $170
D MAnual Controller (SCRG).....($90) $85
Switch-a-Slot (SCRG).....($190) $175
Extend-a-Slot (SCRG).....($35) $32
PROmGRAMER (SCRG).....($149.50) $140
Write Guard Disk Mod Kit (Mark IV).....$40

Books, Books, Books.....compare our discount prices!
"Inside the Apple //e", Little.....($19.95) $18
"Apple II+/IIe Troubleshooting & Repair Guide", Brenner..($19.95) $18
"Apple][Circuit Description", Gayler.....($22.95) $21
"Understanding the Apple II", Sather.....($22.95) $21
"Understanding the Apple //e", Sather (avail. April, price about $21)
"Enhancing Your Apple II, vol. 1", Lancaster.....($15.95) $15
Second edition, with //e information.
"Enhancing Your Apple II/IIe, vol 2", Lancaster ....(May, $19.95) $19
"Assembly Cookbook for the Apple II/IIe", Lancaster.....($21.95) $20
"Incredible Secret Money Machine", Lancaster.....($7.95) $7
"Beneath Apple DOS", Worth & Lechner.....($19.95) $18
"Beneath Apple ProDOS", Worth & Lechner.....($19.95) $18
"What's Where in the Apple", Second Edition.....($19.95) $19
"6502 Assembly Language Programming", Leventhal.....($18.95) $18
"6502 Subroutines", Leventhal.....($18.95) $18
"Real Time Programming -- Neglected Topics", Foster.....($9.95) $9
"Microcomputer Graphics", Myers.....($12.95) $12
"Apple II Assembly Language", DeJong.....($15.95) $15
"Apple II Applications" (Interfacing & I/O experiments)..($13.95) $13
"Apple Programmer's Handbook", Paul Irwin.....($22.95) $21
"Electronically Hearing: Computer Speech Recognition.....($13.95) $13
"Electron. Speaking: Computer Speech Generation (Cater)...($14.95) $14
"Assem. Lang. for Applesoft Programmers", Finley & Myers..($16.95) $16

Add $1.50 per book for US shipping. Foreign orders add postage needed.

Texas residents please add 6 1/8 % sales tax to all orders.

*** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
*** (214) 324-2050 ***
*** We accept Master Card, VISA and American Express ***

plus sign in the picture will also be printed. All other positions of the field will be printed as underlines. Once the field has been displayed in this fashion, DP18 will check the current value in the variable corresponding with the field. If the current value is zero, DP18 merely waits for you to enter digits. If the current value is non-zero, that value is displayed in the field on the screen, to be used as a default value.

When INPUT$ is waiting for you to enter a numeric value, you can type the RETURN key to accept the default value. If no default value is displayed and you type the RETURN key, you will be entering a value of zero. If you begin to type digits, they will enter the field from the right end in "calculator style". Using backspace will cause the displayed value to be popped to the right, deleting the last digit you typed. One digit will be deleted each time you type backspace.

If you type a period, enough zeroes will be automatically entered to reach the displayed decimal point. This makes the digits you typed before the period into an integer. Then as you continue to type digits they will be appended after the decimal point. If you type more fractional digits than can be seen in the displayed field, they do become part of the input value; you just cannot see them on the screen. The value on the screen is rounded up if necessary.

A control-X will erase everything you have typed in the current field and allow you to start over. A control-C will immediately BREAK, stopping the program.

If you type a backspace when there are no digits remaining in a field, DP18 will attempt to go back to the previous field in the same picture. This will only work if the screen has not scrolled during the development of the picture, and requires a little bit of planning. (Isn't that what programming is all about?)

Probably it is time for an example.

```
100 &DP: INPUT $ "HV>>'ENTER X:  '###.#/
    'ENTER Y:  '###.#",X(0),Y(0)
```

Remember how to read pictures from last month's article? The "H" all by itself sets the horizontal cursor position to 0 (beginning of the line). Likewise, "V" sets us to the top line. The ">>" clears from cursor to end of screen. Therefore the "HV>>" does the same thing as a normal HOME command, but from within a picture. The string between apostrophes is printed on the screen. Then "###.#" defines a numeric field, corresponding to the variable X(0). The "/" causes a carriage return to be displayed, and then "ENTER Y:" and the second field.

During execution you will first see the screen clear and the top line become "ENTER X: ___._" followed by a flashing cursor. You can type digits, a sign, a decimal point, backspace, and so on. When you finally type the RETURN a

second line will appear: "ENTER Y: ___._". If you then type a backspace, the cursor will move back to the first line, displaying as a default value whatever you left in that line.

And what about string fields in the INPUT$ command? Again, underlines will be displayed for each position of the string field. If the string already is non-null, its current value will be displayed as a default.

The code that follows is, as has been our practice throughout the DP18 series, preceded by some .EQ lines to define routines previously published, or part of the Apple ROMs. Variable storage is also defined. In the integrated source all these definitions are only done once, and the whole program is assembled together.

When the main execution loop of DP18 encounters the INPUT token, we land at line 1840. Lines 1850-1860 get the character following INPUT, and abort with SYNTAX ERROR if that character is a colon or end-of-line token. Lines 1870-1910 handle INPUT$, by merely loading up zero in the A-register and jumping to PRINT.INPUT (which was listed last month as part of the PRINT USING code). The zero value will be stored in a flag, indicating to PRINT.INPUT later on that it was called from INPUT$ rather than PRINT$. When the picture processor encounters a numeric or string field description in the picture either INPUT.NUM or INPUT.STR will be called, rather than PRINT.NUM or PRINT.STR.

Lines 1930-2510 handle the normal INPUT and INPUT# modes. The character which follows INPUT is stored at INPUT.TYPE, to be checked later. If that character was "#", line 1960 gets the next character to position properly for scanning optional quote or the variable name. Lines 1970-2120 process the optional quote. If it is not there, a "?" prompt is used; it it is there, the string itself is printed. Lines 2090-2110 make a ";" optional after the quote. Normal Applesoft INPUT requires a semicolon after the quote, but DP18's INPUT makes it optional. In fact, you could even get by with a whole bunch of semicolons, if you feel like it....

Lines 2140-2190 read a line of text. If the first character of the line is a control-C, we abort just like Applesoft. An empty line returns a zero value, using line 2500-2510.

Lines 2210-2270 set up the input line, which begins at $0200 (WBUF), so that it can be scanned using CHRGET, after pushing current TXTPTR value on the stack. If the INPUT.TYPE was "#", AS.PARSE and DP.EVALUATE convert the expression down to a value. If not, FIN converts the number string to a value. I could have used PARSE and EVALUATE regardless, but it would take a lot more time to convert plain numbers that way. Lines 2400-2430 restore the old value of TXTPTR, so that we can continue scanning the program.

Lines 2440-2480 scan the input variable name, and store the converted value in that variable. Then back to DP18's main loop to get the next command!

If we are processing an INPUT$ statement, chances are good that we will input a number. If so, the picture processor will call on INPUT.NUM at line 2530. WBUF at this time holds the image of the numeric field description, as amplified from the picture. Lines 2540-2600 copy it into IBUF, because we are going to clobber the WBUF version everytime we re-display the value being entered. IBUF is currently assembled as a 256 byte buffer, which is quite extravagant. Probably this is an area where things could be tightened up, if you need the memory space.

The code beyond line 2530 is hard to follow. I am reminded of the original Adventure game, and its twisty little passages, little twisty passages, and so on. I am going to give it a broad brush, and those of you with an intense interest can explore in more detail on your own.

As each digit is typed, it is appended to the numeric value by ACCUMULATE.DIGIT. Then, after refreshing the picture of the field from IBUF, the value is reconverted to display format and shown on the screen. It may sound inefficient, but it all works nicely. Trimming off digits when backspace is typed is done by truncating the DP18 value and then redisplaying.

LAST.FLD is the routine that tries to back up input to a previous field when you type backspace beyond the first digit. At the beginning of each field, all the necessary parameters are pushed on DP18's stack. LAST.FLD pops these back to move to a previous field. Guess what ... I forgot to check for stack overflow in the STACK.IT subroutine. Should be no problem, however, because only five bytes are stacked for each field, there is room for 24 fields. Since a picture must necessarily be less than 256 characters (maximum length of an Applesoft string) thereby limiting the number of fields, it is unlikely that you will have more than 24 fields stack up. If you think it important to have more, you had better increase the size of STACK.

String input is handled in an analogous fashion by INPUT.STR, starting at line 4970.

As I mentioned before, this is my final article on DP18. But maybe not, if you want more. Some of you might send improvements, corrections, or whatever, and I might pass them along in these pages.

DP18 works, and works well; we're proud of it. You can use DP18 in your programs, even those you plan to sell. Just give us credit where appropriate in your documentation. Remember, you can get all the source code already typed in and integrated together from us for only $50.

```
                    1000 *SAVE S.DP18 INPUT
                    1010 *--------------------------------
                    1020 *     APPLESOFT SUBROUTINES
                    1030 *--------------------------------
D52E-               1040 AS.INLIN      .EQ $D52E    READ A LINE
D559-               1050 AS.PARSE      .EQ $D559    PARSE INPUT BUFFER
D863-               1060 AS.BREAK      .EQ $D863    CTRL-C BREAK
D998-               1070 AS.ADDON      .EQ $D998    ADD (Y) TO TXTPTR
DB5C-               1080 AS.COUT       .EQ $DB5C     PRINT A CHARACTER
DEBE-               1090 AS.CHKCOM     .EQ $DEBE     CHECK FOR COMMA
DEC9-               1100 AS.SYNERR     .EQ $DEC9     SYNTAX ERROR
E452-               1110 AS.GETSPA     .EQ $E452
E5E2-               1120 AS.MOVSTR     .EQ $E5E2
                    1130 *--------------------------------
                    1140 *   MONITOR SUBROUTINES
                    1150 *--------------------------------
FD0C-               1160 MON.RDKEY     .EQ $FD0C
FC66-               1170 MON.LF        .EQ $FC66
                    1180 *--------------------------------
                    1190 *       DP SUBROUTINES PRINTED ELSEWHERE
                    1200 *--------------------------------
FFFF-               1210 DP.NEXT.CMD           .EQ $FFFF
FFFF-               1220 DP.EVALUATE           .EQ $FFFF
FFFF-               1230 MOVE.DAC.YA           .EQ $FFFF
FFFF-               1240 DP.VTAB               .EQ $FFFF
FFFF-               1250 DP.INT                .EQ $FFFF
FFFF-               1260 DP.FALSE              .EQ $FFFF
FFFF-               1270 MOVE.DAC.TEMP1        .EQ $FFFF
FFFF-               1280 MOVE.TEMP1.DAC        .EQ $FFFF
FFFF-               1290 PRINT.INPUT           .EQ $FFFF
FFFF-               1300 FIN                   .EQ $FFFF
FFFF-               1310 GET.A.VAR             .EQ $FFFF
FFFF-               1320 CHECK.DP.VAR          .EQ $FFFF
FFFF-               1330 MOVE.YA.DAC           .EQ $FFFF
FFFF-               1340 PRUS.CLEAR            .EQ $FFFF
FFFF-               1350 PRUS.NEXT             .EQ $FFFF
FFFF-               1360 ACCUMULATE.DIGIT      .EQ $FFFF
FFFF-               1370 PRT.NUM.1             .EQ $FFFF
FFFF-               1380 PRINT.STR.1           .EQ $FFFF
                    1390 *--------------------------------
                    1400 *       PAGE ZERO USAGE
                    1410 *--------------------------------
11-                 1420 AS.VALTYP     .EQ $11
21-                 1430 MON.WNDWIDTH  .EQ $21
24-                 1440 MON.CH        .EQ $24
25-                 1450 MON.CV        .EQ $25
71-                 1460 AS.FRESPA     .EQ $71,72
B1-                 1470 AS.CHRGET     .EQ $B1
B7-                 1480 AS.CHRGOT     .EQ $B7
B8-                 1490 TXTPTR        .EQ $B8,B9
F9-                 1500 P2            .EQ $F9
FD-                 1510 P1            .EQ $FD       GP POINTER
                    1520 *--------------------------------
0200-               1530 WBUF          .EQ $0200
                    1540 *--------------------------------
                    1550 *       WORK AREAS FOR DPFP
                    1560 *--------------------------------
0800-               1570 DECFLG                .BS 1
0801-               1580 DAC.EXPONENT          .BS 1
0802-               1590 DAC.SIGN              .BS 1
0803-               1600 IBUF                  .BS 256
0903-               1610 STACK.PNTR            .BS 1
0904-               1620 STACK                 .BS 12*10
097C-               1630 W                     .BS 1
097D-               1640 D                     .BS 1
097E-               1650 OLD.W                 .BS 1
097F-               1660 OLD.D                 .BS 1
0980-               1670 DGTCNT                .BS 1
0981-               1680 INPUT.TYPE            .BS 1
0982-               1690 FOUND.NUM             .BS 1
0983-               1700 FOUND.STR             .BS 1
0984-               1710 FOUND.LEN             .BS 1
0985-               1720 FOUND.CHAR            .BS 1
0986-               1730 FILL.CHAR             .BS 1
0987-               1740 ZERO.CHAR             .BS 1
```

```
0988-                1750 FLD.FLAG             .BS 1
0989-                1760 FLD.START            .BS 1
098A-                1770 TEMP                 .BS 2
098C-                1780 RESULT               .BS 2
098E-                1790 DEFAULT.FLAG         .BS 1
098F-                1800 LEN                  .BS 1
                     1810 *--------------------------------
0990- 4C C9 DE       1820 DP.SYN3 JMP AS.SYNERR
                     1830 *--------------------------------
                     1840 DP.INPUT
0993- 20 B1 00       1850         JSR AS.CHRGET
0996- F0 F8          1860         BEQ DP.SYN3  ...COLON OR EOL
                     1870 *---INPUT USING-----------------
0998- C9 24          1880         CMP #'$'     INPUT USING PICTURE?
099A- D0 05          1890         BNE .1       ...NO
099C- A9 00          1900         LDA #0       ...YES, SIGNAL "INPUT" AND JOIN
099E- 4C FF FF       1910         JMP PRINT.INPUT      "PRINT $"
                     1920 *---INPUT AN EXPRESSION----------
09A1- 8D 81 09       1930 .1      STA INPUT.TYPE    ="#" IF EXP, ELSE <>"#"
09A4- C9 23          1940         CMP #'#'          INPUT AN EXPRESSION?
09A6- D0 03          1950         BNE .2            ...NO
09A8- 20 B1 00       1960         JSR AS.CHRGET     ...YES, GET NEXT CHAR
09AB- A2 BF          1970 .2      LDX #"?"     PROMPT CHAR FOR NO QUOTE
09AD- C9 22          1980         CMP #'"'     QUOTE?
09AF- D0 1C          1990         BNE .6       ...NO, SIMPLE INPUT
09B1- A0 00          2000         LDY #0       ...YES, PRINT IT NOW
09B3- C8             2010 .3      INY
09B4- B1 B8          2020         LDA (TXTPTR),Y    NEXT QUOTED CHARACTER
09B6- F0 D8          2030         BEQ DP.SYN3       ...NO CLOSING QUOTE
09B8- C9 22          2040         CMP #'"'          CLOSING QUOTE YET?
09BA- F0 05          2050         BEQ .4            ...YES
09BC- 20 5C DB       2060         JSR AS.COUT       ...NO, PRINT CHARACTER
09BF- D0 F2          2070         BNE .3            ...ALWAYS
09C1- 20 98 D9       2080 .4      JSR AS.ADDON      ADD (Y) TO TXTPTR
09C4- 20 B1 00       2090 .5      JSR AS.CHRGET     SCAN NEXT CHAR
09C7- C9 3B          2100         CMP #';'          ALLOW OPTIONAL SEMICOLON
09C9- F0 F9          2110         BEQ .5            ...KEEP LOOKING TILL NOT ';'
09CB- A2 80          2120         LDX #$80          NULL PROMPT CHARACTER
                     2130 *---READ A LINE OF TEXT----------
09CD- 20 2E D5       2140 .6      JSR AS.INLIN      '?' OR NULL PROMPT
09D0- AD 00 02       2150         LDA WBUF          CHECK FOR EMPTY LINE
09D3- F0 47          2160         BEQ .11           ...EMPTY LINE
09D5- C9 03          2170         CMP #$03          CTRL-C?
09D7- D0 03          2180         BNE .7            ...NO
09D9- 4C 63 D8       2190         JMP AS.BREAK      ABORT INPUT
                     2200 *---PARSE THE INPUT LINE---------
09DC- A5 B8          2210 .7      LDA TXTPTR        SAVE TXTPTR, WHICH POINTS
09DE- 48             2220         PHA               AT THE PROGRAM
09DF- A5 B9          2230         LDA TXTPTR+1
09E1- 48             2240         PHA
09E2- 86 B8          2250         STX TXTPTR        MAKE TXTPTR POINT AT INPUT BUFFER
09E4- 84 B9          2260         STY TXTPTR+1
09E6- 20 B1 00       2270         JSR AS.CHRGET     GET FIRST CHAR FROM LINE
09E9- AC 81 09       2280         LDY INPUT.TYPE    SEE IF SIMPLE OR EXPRESSIONS
09EC- C0 23          2290         CPY #'#'
09EE- D0 14          2300         BNE .8            SIMPLE NUMERIC INPUT
09F0- 20 59 D5       2310         JSR AS.PARSE      EXPRESSION INPUT, SO PARSE
09F3- A9 FF          2320         LDA #WBUF-1       POINT AT INPUT BUFFER AGAIN
09F5- 85 B8          2330         STA TXTPTR        SO EVALUATE CAN PROCESS THE
09F7- A9 01          2340         LDA /WBUF-1       PARSED LINE
09F9- 85 B9          2350         STA TXTPTR+1
09FB- 20 B1 00       2360         JSR AS.CHRGET     SCAN FIRST CHAR
09FE- 20 FF FF       2370         JSR DP.EVALUATE   EVALUATE THE EXPRESSION
0A01- 4C 07 0A       2380         JMP .9
0A04- 20 FF FF       2390 .8      JSR FIN           SIMPLE NUMERIC INPUT
0A07- 68             2400 .9      PLA               RESTORE TXTPTR TO PROGRAM
0A08- 85 B9          2410         STA TXTPTR+1
0A0A- 68             2420         PLA
0A0B- 85 B8          2430         STA TXTPTR
0A0D- 20 B7 00       2440 .10     JSR AS.CHRGOT     GET CURRENT PROGRAM CHAR
0A10- 20 FF FF       2450         JSR GET.A.VAR     GET INPUT VARIABLE
0A13- 20 FF FF       2460         JSR CHECK.DP.VAR  MUST BE DP18 VARIABLE
0A16- 20 FF FF       2470         JSR MOVE.DAC.YA   STORE INPUT VALUE
0A19- 4C FF FF       2480         JMP DP.NEXT.CMD   ...FINISHED?
```

```
                        2490 *---EMPTY INPUT LINE-------------
0A1C- 20 FF FF          2500 .11    JSR DP.FALSE       RETURN VALUE = 0
0A1F- 4C 0D 0A          2510        JMP .10
                        2520 *--------------------------------
                        2530 INPUT.NUM
0A22- A9 00             2540        LDA #0         TERMINATE STRING IN BUFFERS
0A24- 9D 03 08          2550        STA IBUF,X
0A27- 9D 00 02          2560        STA WBUF,X
0A2A- BD FF 01          2570 .1     LDA WBUF-1,X       COPY STRING TO IBUF
0A2D- 9D 02 08          2580        STA IBUF-1,X
0A30- CA                2590        DEX
0A31- D0 F7             2600        BNE .1
0A33- AD 86 09          2610        LDA FILL.CHAR
0A36- 8D 8A 09          2620        STA TEMP
0A39- 20 EC 0B          2630        JSR STACK.IT
0A3C- 20 BE DE          2640        JSR AS.CHKCOM      MUST HAVE COMMA
0A3F- 20 FF FF          2650        JSR GET.A.VAR
0A42- 20 FF FF          2660        JSR CHECK.DP.VAR
0A45- 8D 8C 09          2670        STA RESULT    SAVE ADR OF VARIABLE
0A48- 8C 8D 09          2680        STY RESULT+1
0A4B- 20 FF FF          2690        JSR MOVE.YA.DAC  MOVE DEFAULT INTO DAC
0A4E- AD 7C 09          2700        LDA W
0A51- 8D 7E 09          2710        STA OLD.W
0A54- A9 01             2720        LDA #1
0A56- 8D 8E 09          2730        STA DEFAULT.FLAG
0A59- AD 01 08          2740        LDA DAC.EXPONENT  IS DAC 0?
0A5C- D0 03             2750        BNE INP.X1    NO
0A5E- 20 9B 0B          2760 INP.X  JSR INP.ZERO.DAC DEFAULT IS 0 OR CTRL-X
0A61- A9 00             2770 INP.X1 LDA #0
0A63- 8D 88 09          2780        STA FLD.FLAG
0A66- 8D 80 09          2790        STA DGTCNT
0A69- 8D 00 08          2800        STA DECFLG
0A6C- AD 7D 09          2810        LDA D
0A6F- 8D 7F 09          2820        STA OLD.D
0A72- A9 5F             2830        LDA #$5F      UNDERLINE
0A74- 8D 86 09          2840        STA FILL.CHAR
                        2850 INP.NEXT.ZERO.CHAR
0A77- 8D 87 09          2860        STA ZERO.CHAR
                        2870 *--------------------------------
                        2880 INP.NEXT
0A7A- 20 72 0B          2890        JSR INP.PRINT.NUM PRINT THE NUMBER
0A7D- 20 FF FF          2900        JSR MOVE.TEMP1.DAC
0A80- 20 0C FD          2910        JSR MON.RDKEY
0A83- 29 7F             2920        AND #$7F
0A85- C9 0D             2930        CMP #$0D      RETURN?
0A87- F0 1A             2940        BEQ .2             ...YES
0A89- AE 8E 09          2950        LDX DEFAULT.FLAG
0A8C- F0 07             2960        BEQ .1        NO DEFAULT
0A8E- 20 9B 0B          2970        JSR INP.ZERO.DAC IGNORE DEFAULT
0A91- C9 08             2980        CMP #8        BACKSPACE?
0A93- F0 E5             2990        BEQ INP.NEXT YES,IGNORE
                        3000 *---DIGIT------------------------
0A95- C9 30             3010 .1     CMP #'0       SEE IF NUMBER
0A97- 90 2C             3020        BCC .4        NO
0A99- C9 3A             3030        CMP #'9+1
0A9B- B0 28             3040        BCS .4        NO
0A9D- 20 FF FF          3050        JSR ACCUMULATE.DIGIT
0AA0- 4C 7A 0A          3060        JMP INP.NEXT
                        3070 *---CARRIAGE RETURN--------------
0AA3- AD 80 09          3080 .2     LDA DGTCNT    IS NUMBER 0?
0AA6- 0D 8E 09          3090        ORA DEFAULT.FLAG
0AA9- D0 03             3100        BNE .3        NO
0AAB- 8D 01 08          3110        STA DAC.EXPONENT YES,SO ZERO THE EXPONENT
0AAE- AD 8C 09          3120 .3     LDA RESULT    GET ADR OF VAR
0AB1- AC 8D 09          3130        LDY RESULT+1
0AB4- 20 FF FF          3140        JSR MOVE.DAC.YA  PUT IT IN VAR
0AB7- AD 8A 09          3150        LDA TEMP      RESTORE ORIGINAL FILL CHAR
0ABA- 8D 86 09          3160        STA FILL.CHAR
0ABD- A9 30             3170        LDA #'0
0ABF- 8D 87 09          3180        STA ZERO.CHAR
0AC2- 4C 72 0B          3190        JMP INP.PRINT.NUM PRINT THE NUMBER
                        3200 *                          AND RETURN
                        3210 *---DECIMAL POINT----------------
0AC5- C9 2E             3220 .4     CMP #'.       DEC POINT?
0AC7- D0 15             3230        BNE .5        ...NO
```

```
                      3240 *       SEC           'CMP' LEFT CARRY SET
0AC9- 6E 00 08        3250         ROR DECFLG    FOUND DEC PT
0ACC- 2C 00 08        3260         BIT DECFLG
0ACF- 70 A9           3270         BVS INP.NEXT        TWO DEC PTS.
0AD1- A9 40           3280         LDA #$40
0AD3- 18              3290         CLC
0AD4- 6D 80 09        3300         ADC DGTCNT
0AD7- 8D 01 08        3310         STA DAC.EXPONENT
0ADA- A9 30           3320         LDA #'0
0ADC- F0 99           3330         BEQ INP.NEXT.ZERO.CHAR  ALWAYS
                      3340 *---MINUS SIGN--------------------
0ADE- C9 2D           3350 .5      CMP #'-       MINUS?
0AE0- D0 05           3360         BNE .6
                      3370 *       SEC           'CMP' LEFT CARRY SET
0AE2- 6E 02 08        3380         ROR DAC.SIGN MAKE DAC NEGATIVE
0AE5- D0 93           3390         BNE INP.NEXT ...ALWAYS
                      3400 *---PLUS SIGN---------------------
0AE7- C9 2B           3410 .6      CMP #'+       PLUS?
0AE9- D0 05           3420         BNE .7        ...NO
0AEB- 8D 02 08        3430         STA DAC.SIGN PUT POSITIVE VALUE IN SIGN
0AEE- F0 8A           3440         BEQ INP.NEXT ...ALWAYS
                      3450 *---CTRL-X------------------------
0AF0- C9 18           3460 .7      CMP #$18      CTRL-X?
0AF2- D0 09           3470         BNE .8
0AF4- AD 7F 09        3480         LDA OLD.D
0AF7- 8D 7D 09        3490         STA D
0AFA- 4C 5E 0A        3500         JMP INP.X
                      3510 *---CTRL-C------------------------
0AFD- C9 03           3520 .8      CMP #$3       CTRL-C?
0AFF- D0 03           3530         BNE .9        ...NO, TRY BACKSPACE
0B01- 4C 63 D8        3540         JMP AS.BREAK
                      3550 *---BACKSPACE---------------------
0B04- C9 08           3560 .9      CMP #$08      BACKSPACE?
0B06- D0 67           3570         BNE .17       ...NO, TAKE PATH TO INP.NEXT
0B08- AD 00 08        3580         LDA DECFLG
0B0B- 10 0B           3590         BPL .10
0B0D- AD 01 08        3600         LDA DAC.EXPONENT
0B10- 38              3610         SEC
0B11- E9 40           3620         SBC #$40
0B13- CD 80 09        3630         CMP DGTCNT
0B16- F0 39           3640         BEQ .15       REMOVE DEC PT ONLY
                      3650 *-------------------------------
0B18- AD 01 08        3660 .10     LDA DAC.EXPONENT
0B1B- 48              3670         PHA           SAVE EXPONENT
0B1C- AD 80 09        3680         LDA DGTCNT
0B1F- 18              3690         CLC
0B20- 69 3F           3700         ADC #$3F
0B22- 8D 01 08        3710         STA DAC.EXPONENT
0B25- 20 FF FF        3720         JSR DP.INT    CHOP OFF LAST DIGIT
0B28- AD 01 08        3730         LDA DAC.EXPONENT
0B2B- F0 1E           3740         BEQ .14       THE NUMBER IS 0, SO RESET EVERYTHING
0B2D- 68              3750 .11     PLA
0B2E- 8D 01 08        3760         STA DAC.EXPONENT
0B31- AD 80 09        3770         LDA DGTCNT
0B34- D0 06           3780         BNE .12
0B36- 20 AF 0B        3790         JSR LAST.FLD
0B39- 4C 7A 0A        3800         JMP INP.NEXT
0B3C- CE 80 09        3810 .12     DEC DGTCNT
0B3F- D0 03           3820         BNE .13
0B41- CE 01 08        3830         DEC DAC.EXPONENT
0B44- AD 00 08        3840 .13     LDA DECFLG
0B47- 10 1E           3850         BPL .16       DELETE BY SHIFT
0B49- 30 24           3860         BMI .17       ALWAYS
                      3870 *-------------------------------
0B4B- AD 00 08        3880 .14     LDA DECFLG
0B4E- 10 DD           3890         BPL .11
0B50- 68              3900         PLA
0B51- A9 3F           3910 .15     LDA #$3F
0B53- 38              3920         SEC
0B54- ED 7F 09        3930         SBC OLD.D
0B57- 6D 80 09        3940         ADC DGTCNT
0B5A- 8D 01 08        3950         STA DAC.EXPONENT
0B5D- A9 00           3960         LDA #0
0B5F- 8D 00 08        3970         STA DECFLG
0B62- A9 5F           3980         LDA #$5F
0B64- 4C 77 0A        3990         JMP INP.NEXT.ZERO.CHAR
                      4000 *-------------------------------
```

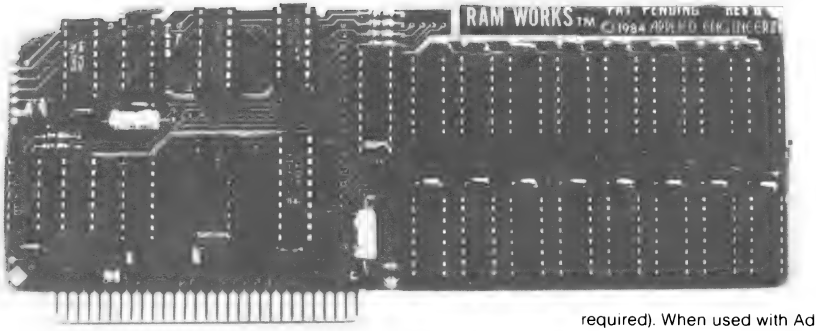
```
0B67- AD 80 09 4010 .16     LDA DGTCNT
0B6A- F0 03    4020         BEQ .17
0B6C- CE 01 08 4030         DEC DAC.EXPONENT
0B6F- 4C 7A 0A 4040 .17     JMP INP.NEXT
               4050 *-------------------------------
               4060 INP.PRINT.NUM
0B72- A2 FF    4070         LDX #-1       COPY IBUF TO WBUF
0B74- E8       4080 .1      INX
0B75- BD 03 08 4090         LDA IBUF,X
0B78- 9D 00 02 4100         STA WBUF,X
0B7B- D0 F7    4110         BNE .1
0B7D- 20 20 0D 4120         JSR RESTORE.HV.FROM.STACK
0B80- AD 7E 09 4130         LDA OLD.W
0B83- 8D 7C 09 4140         STA W
0B86- AD 7F 09 4150         LDA OLD.D
0B89- 8D 7D 09 4160         STA D
0B8C- 20 FF FF 4170         JSR MOVE.DAC.TEMP1
0B8F- AD 00 08 4180         LDA DECFLG
0B92- 48       4190         PHA
0B93- 20 FF FF 4200         JSR PRT.NUM.1
0B96- 68       4210         PLA
0B97- 8D 00 08 4220         STA DECFLG
0B9A- 60       4230         RTS
               4240 *-------------------------------
               4250 INP.ZERO.DAC
0B9B- 48       4260         PHA
0B9C- 20 FF FF 4270         JSR DP.FALSE PUT 0 IN DAC
0B9F- A9 40    4280         LDA #$40
0BA1- 38       4290         SEC
0BA2- ED 7D 09 4300         SBC D         CALCULATE EXPONENT
0BA5- 8D 01 08 4310         STA DAC.EXPONENT
0BA8- A9 00    4320         LDA #0
0BAA- 8D 8E 09 4330         STA DEFAULT.FLAG
0BAD- 68       4340         PLA
0BAE- 60       4350         RTS
               4360 *-------------------------------
               4370 LAST.FLD
0BAF- AC 03 09 4380         LDY STACK.PNTR
0BB2- 88       4390         DEY
0BB3- 88       4400         DEY
0BB4- 88       4410         DEY
0BB5- 88       4420         DEY
0BB6- 88       4430         DEY
0BB7- D0 01    4440         BNE .1
0BB9- 60       4450         RTS           FIRST FIELD
0BBA- 68       4460 .1      PLA           DISCARD JSR LAST.FLD
0BBB- 68       4470         PLA                 "
0BBC- 68       4480         PLA           DISCARD JSR INPUT.NUM
0BBD- 68       4490         PLA                 "
0BBE- 68       4500         PLA           DISCARD Y-REG
0BBF- 68       4510         PLA           DISCARD JSR PRT.NUM.IF.NEEDED
0BC0- 68       4520         PLA                 "
0BC1- 68       4530         PLA           DISCARD JSR LOOKUP
0BC2- 68       4540         PLA                 "
0BC3- 88       4550         DEY
0BC4- B9 04 09 4560         LDA STACK,Y
0BC7- 85 B9    4570         STA TXTPTR+1
0BC9- 88       4580         DEY
0BCA- B9 04 09 4590         LDA STACK,Y
0BCD- 85 B8    4600         STA TXTPTR
0BCF- 88       4610         DEY
0BD0- B9 04 09 4620         LDA STACK,Y
0BD3- 48       4630         PHA           SAVE INDEX INTO PICTURE
0BD4- 88       4640         DEY
0BD5- B9 04 09 4650         LDA STACK,Y
0BD8- 20 FF FF 4660         JSR DP.VTAB
0BDB- 88       4670         DEY
0BDC- B9 04 09 4680         LDA STACK,Y
0BDF- 85 24    4690         STA MON.CH
0BE1- 8C 03 09 4700         STY STACK.PNTR
0BE4- 68       4710         PLA           RESTORE INDEX INTO PICTURE
0BE5- A8       4720         TAY
0BE6- 20 FF FF 4730         JSR PRUS.CLEAR
0BE9- 4C FF FF 4740         JMP PRUS.NEXT
               4750 *-------------------------------
```

```
                        4760 STACK.IT
0BEC- AC 03 09  4770            LDY STACK.PNTR
0BEF- A5 24     4780            LDA MON.CH    SAVE WHERE THE FIELD IS
0BF1- 99 04 09  4790            STA STACK,Y
0BF4- C8        4800            INY
0BF5- A5 25     4810            LDA MON.CV
0BF7- 99 04 09  4820            STA STACK,Y
0BFA- C8        4830            INY
0BFB- CE 89 09  4840            DEC FLD.START
0BFE- AD 89 09  4850            LDA FLD.START
0C01- 99 04 09  4860            STA STACK,Y
0C04- C8        4870            INY
0C05- A5 B8     4880            LDA TXTPTR
0C07- 99 04 09  4890            STA STACK,Y  SAVE TXTPTR
0C0A- C8        4900            INY
0C0B- A5 B9     4910            LDA TXTPTR+1
0C0D- 99 04 09  4920            STA STACK,Y
0C10- C8        4930            INY
0C11- 8C 03 09  4940            STY STACK.PNTR
0C14- 60        4950            RTS
                        4960 *--------------------------------
                        4970 INPUT.STR
0C15- 20 EC 0B  4980            JSR STACK.IT
0C18- 20 BE DE  4990            JSR AS.CHKCOM   MUST HAVE COMMA
0C1B- 20 FF FF  5000            JSR GET.A.VAR   GET ADR OF VAR
0C1E- A6 11     5010            LDX AS.VALTYP   STR OR NUM
0C20- 30 03     5020            BMI .1          OK
0C22- 4C C9 DE  5030            JMP AS.SYNERR   MUST BE STRING
0C25- 85 FD     5040 .1         STA P1
0C27- 84 FE     5050            STY P1+1
0C29- A0 00     5060            LDY #0          GET STRING
0C2B- 8C 8E 09  5070            STY DEFAULT.FLAG
0C2E- 8C 88 09  5080            STY FLD.FLAG
0C31- 8C 8F 09  5090            STY LEN
0C34- B1 FD     5100            LDA (P1),Y      LENGTH
0C36- F0 22     5110            BEQ .3          NULL STRING, SO DO NOTHING
0C38- 8D 8F 09  5120            STA LEN
0C3B- C8        5130            INY
0C3C- B1 FD     5140            LDA (P1),Y      ADR OF STRING
0C3E- 85 F9     5150            STA P2          LO ADR
0C40- C8        5160            INY
0C41- B1 FD     5170            LDA (P1),Y
0C43- 85 FA     5180            STA P2+1        HI ADR
0C45- AC 8F 09  5190            LDY LEN         GET LENGTH
0C48- 88        5200            DEY
0C49- B1 F9     5210 .2         LDA (P2),Y
0C4B- 99 00 02  5220            STA WBUF,Y
0C4E- 88        5230            DEY
0C4F- D0 F8     5240            BNE .2
0C51- B1 F9     5250            LDA (P2),Y      MOVE LAST BYTE
0C53- 8D 00 02  5260            STA WBUF
0C56- C8        5270            INY             Y = 1
0C57- 8D 8E 09  5280            STA DEFAULT.FLAG  YES THERE IS A DEFAULT
0C5A- A9 00     5290 .3         LDA #WBUF
0C5C- 85 F9     5300            STA P2
0C5E- A9 02     5310            LDA /WBUF
0C60- 85 FA     5320            STA P2+1
0C62- D0 05     5330            BNE IS.X1       ALWAYS
                        5340 *--------------------------------
0C64- A9 00     5350 IS.X       LDA #0
0C66- 8D 8F 09  5360            STA LEN
0C69- AD 84 09  5370 IS.X1      LDA FOUND.LEN
0C6C- 48        5380            PHA
0C6D- AD 85 09  5390            LDA FOUND.CHAR
0C70- 48        5400            PHA
0C71- 20 20 0D  5410            JSR RESTORE.HV.FROM.STACK
0C74- A9 5F     5420            LDA #$5F        UNDERLINE
0C76- 8D 86 09  5430            STA FILL.CHAR
0C79- AD 8F 09  5440            LDA LEN
0C7C- 20 FF FF  5450            JSR PRINT.STR.1
0C7F- 68        5460            PLA
0C80- 8D 85 09  5470            STA FOUND.CHAR
0C83- 68        5480            PLA
0C84- 8D 84 09  5490            STA FOUND.LEN
0C87- CD 8F 09  5500            CMP LEN
0C8A- 90 17     5510            BCC .3          OVERFLOW
```

```
                        5520 *---FIND END OF STRING & PUT CURSOR THERE---
0C8C- 20 20 0D          5530         JSR RESTORE.HV.FROM.STACK
0C8F- 18                5540         CLC
0C90- 6D 8F 09          5550         ADC LEN        ADD LENGTH OF STRING
0C93- C5 21             5560 .1      CMP MON.WNDWIDTH LONGER THAN WINDOW?
0C95- 90 0A             5570         BCC .2
0C97- E5 21             5580         SBC MON.WNDWIDTH WRAP AROUND
0C99- 48                5590         PHA
0C9A- 20 66 FC          5600         JSR MON.LF    JUMP DOWN TO NEXT LINE
0C9D- 68                5610         PLA
0C9E- 4C 93 0C          5620         JMP .1
0CA1- 85 24             5630 .2      STA MON.CH    PUT COLUMN BACK IN CH
                        5640 *---INPUT A CHAR NOW-------------
0CA3- 20 0C FD          5650 .3      JSR MON.RDKEY
0CA6- 29 7F             5660         AND #$7F
                        5670 *---CARRIAGE RETURN--------------
0CA8- C9 0D             5680         CMP #$0D       RETURN?
0CAA- D0 31             5690         BNE .5         ...NO
0CAC- AD 8E 09          5700         LDA DEFAULT.FLAG
0CAF- D0 1E             5710         BNE .4         DEFAULT, SO LEAVE IT ALONE
0CB1- AD 8F 09          5720         LDA LEN        GET LENGTH
0CB4- 20 52 E4          5730         JSR AS.GETSPA MAKE ROOM FOR STRING
0CB7- A0 00             5740         LDY #0         MOVE DATA INTO VARIABLE
0CB9- 91 FD             5750         STA (P1),Y     LENGTH
0CBB- A5 71             5760         LDA AS.FRESPA
0CBD- C8                5770         INY
0CBE- 91 FD             5780         STA (P1),Y     LO ADDRESS
0CC0- A5 72             5790         LDA AS.FRESPA+1
0CC2- C8                5800         INY
0CC3- 91 FD             5810         STA (P1),Y     HI ADDRESS
0CC5- A2 00             5820         LDX #WBUF
0CC7- A0 02             5830         LDY /WBUF
0CC9- AD 8F 09          5840         LDA LEN
0CCC- 20 E2 E5          5850         JSR AS.MOVSTR
0CCF- 20 20 0D          5860 .4      JSR RESTORE.HV.FROM.STACK
0CD2- A9 20             5870         LDA #$20       SPACE
0CD4- 8D 86 09          5880         STA FILL.CHAR
0CD7- AD 8F 09          5890         LDA LEN
0CDA- 4C FF FF          5900         JMP PRINT.STR.1   PRINT IT ONE MORE TIME
                        5910 *--------------------------------
0CDD- AE 8E 09          5920 .5      LDX DEFAULT.FLAG
0CE0- F0 0F             5930         BEQ .6         ...NO DEFAULT
0CE2- A2 00             5940         LDX #0
0CE4- 8E 8E 09          5950         STX DEFAULT.FLAG GET RID OF DEFAULT
0CE7- 8E 8F 09          5960         STX LEN        NULL STRING
0CEA- C9 08             5970         CMP #8         BACKSPACE AND DEFAULT?
0CEC- D0 18             5980         BNE .8
0CEE- 4C 69 0C          5990         JMP IS.X1
                        6000 *---BACKSPACE---------------------
0CF1- C9 08             6010 .6      CMP #8         BACKSPACE?
0CF3- D0 11             6020         BNE .8
0CF5- AD 8F 09          6030         LDA LEN
0CF8- D0 06             6040         BNE .7
0CFA- 20 AF 0B          6050         JSR LAST.FLD BACKUP A FIELD
0CFD- 4C 69 0C          6060         JMP IS.X1
0D00- CE 8F 09          6070 .7      DEC LEN
0D03- 4C 69 0C          6080         JMP IS.X1
                        6090 *---CTRL-X-----------------------
0D06- C9 18             6100 .8      CMP #$18       CTRL-X?
0D08- D0 03             6110         BNE .9
0D0A- 4C 64 0C          6120         JMP IS.X
                        6130 *---CTRL-C-----------------------
0D0D- C9 03             6140 .9      CMP #3         CTRL-C?
0D0F- D0 03             6150         BNE .10        ...NO
0D11- 4C 63 D8          6160         JMP AS.BREAK
                        6170 *---CHAR FOR STRING--------------
0D14- AC 8F 09          6180 .10     LDY LEN        NORMAL CHAR,
0D17- 99 00 02          6190         STA WBUF,Y     SAVE IT
0D1A- EE 8F 09          6200         INC LEN
0D1D- 4C 69 0C          6210         JMP IS.X1
                        6220 *--------------------------------
                        6230 RESTORE.HV.FROM.STACK
0D20- AC 03 09          6240         LDY STACK.PNTR
0D23- B9 00 09          6250         LDA STACK-4,Y
0D26- 20 FF FF          6260         JSR DP.VTAB
0D29- B9 FF 08          6270         LDA STACK-5,Y
0D2C- 85 24             6280         STA MON.CH
0D2E- 60                6290         RTS
```

RAMWORKS™

800K Apple Works Desk Top, 450K Visicalc, 1 Meg Solid State Disk
Double Hi-Res, RGB, Totally Apple Compatible!



RAMWORKS™—A card that plugs into the Apple IIe auxiliary slot and functions EXACTLY like Apple's extended 80 column card (in fact, a 128K RAMWORKS™ actually costs less than Apple's 64K extended card) but with RAMWORKS™ you get more memory, 80 column text, a 3-year warranty and most importantly, room to grow without using more slots. A design so advanced there's a patent pending on it. If you have a IIc or an IBM, we suggest you do what everybody does, trade it in on a IIe.

RAMWORKS™ can be purchased in a wide range of sizes and is user upgradeable using either 64K RAMS or the new 256K RAMS. In fact, RAMWORKS™ is the only auxiliary slot card on the market that will allow the new 256K RAMS to be used. If you already have an extended 80 column card, no problem. Just unplug the 64K RAMS and plug them into the RAMWORKS™ for an additional 64K. A RGB option is also available, you can order it with your RAMWORKS™ card or add it on at a later date.

RAMWORKS™ saves you time, money, slots and hassle. You'll have additional memory NOW and in the future.

Ramworks™	
64K Installed	$ 179
128K Installed	$ 249
256K Installed	$ 399
512K Installed	$ 649
1 MEG Installed	$1199
RGB Option (May be added later)	$ 129

LOW COST SOFTWARE OPTIONS

Ram Drive IIe

Ram Drive IIe will give you a high speed solid state disk drive. The Ram Drive IIe software features audio-visual access indicators, easy setup for turnkey operation, and easy menu driven documentation. The program can be modified and is copyable. If you have a 64K RAMWORKS™ Ram Drive IIe will act as half a disk drive. If you have a 128K or larger RAMWORKS™ Ram Drive IIe will act as a full disk drive. Either way, your programs will load and save over 20 times faster. Ram Drive IIe is compatible with APPLESOFT, PRO DOS, DOS 3.3, and PASCAL. The disk also includes a high speed RAMdisk copying program. Ram Drive is another disk drive only 20 times faster. And no whirring, clicking and waiting! **PRICE $29**

CP/M Ram Drive IIe

CP/M Ram Drive IIe is just like the Ram Drive IIe above, only for CP/M.

CP/M Ram Drive IIe runs on any Z-80 card that runs standard CP/M i.e. Applied Engineering Z-80 Plus or Microsoft Soft Card. CP/M Ram Drive will dramatically speed up the operation of most CP/M software because CP/M normally goes to disk fairly often. Fast acting software like dBase II, Wordstar and Turbo Pascal becomes virtually instantaneous when used with CP/M Ram Drive.

PRICE $29

VC IIe Expander

VC IIe expander gives owners of Visicalc IIe and Advanced Visicalc IIe increased storage. When used with VC IIe you'll get 141K workspace (128K RAMWORKS™ or larger required). When used with Advanced VC IIe you'll get 131K with a 128K RAMWORKS™, 250K with a 256K RAMWORKS™ and 450K with a 512K RAMWORKS™ **PRICE $29**

Apple Works Expand

Although Apple Works is compatible with all sizes of RAMWORKS™, Apple Works only "sees" its first 64K bank giving you a 55K desktop. Our Apple Works expand program will make a modification to Apple Works that simply lets it know you've got more memory, giving you 101K work space. **PRICE $29**

Super Apple Works Expand

This souped-up version of Apple Works expand doesn't stop at a 101K desk top, in fact Super Apple Works Expand figures out how much memory your RAMWORKS™ has to give Apple Works the following desktop sizes:

RAMWORKS™	APPLEWORKS DESKTOP
128K	101K
256K	200K
512K	400K
1 MEG	800K

PRICE $39

AppleWorks can also be put in the RAMWORKS™ card to eliminate disk access, thereby dramatically speeding up the program.

APPLIED ENGINEERING

Send Check or Money Order to
Applied Engineering
P.O. Box 798, Carrollton, TX 75006
Call (214) 492-2027
8 a.m. to 11 p.m. 7 days a week. MasterCard, Visa & C.O.D. Welcome. No extra charge for credit cards. Texas residents add 5 1/2% sales tax. Add $10.00 if outside U.S.A.

Questions and Answers

I have just finished installing Version 2.0 of your assembler, and I have a few questions.

a. First, how is the line length of the escape-L changed? The short line looks ridiculous on an 80-column screen. I would also like to change the first character from "*" to ";".

b. How can I get the assembler to initialize things with DOS's MON CI modes set?

c. In working with big programs, it is easy to exceed line number 9999. It happens all the time. As new lines get added, the formatting of lines around 9999 goes haywire, as the spacing is done according to the line number at the time of entry. Thus when a line number changes from 4 to 5 digits or vice versa due to renumbering the opcode and operand columns no longer line up properly. What can be done about the erratic column alignment?

d. I noticed that the symbol table generated by an assembly takes more memory with version 2.0 than it did with 1.1. Why?

e. There appear to be two errors in the sample program S.INLINE on the Macro 2.0 disk. The comment on how to use it shows a comma between the &INPUT and the string variable, when the program in fact requires that there be NO comma. Then, the first line of the main routine does a CMP, which should be an LDA. With these corrections, the program is great. &INPUT will accept input from keyboard or disk, and reads the complete record including commas, quotes, and colons. This I find rather useful.

Mike Lawrie, South Africa

a. The routine which generates the star-dash line starts at $DB21, with the following:

```
TXA
BEQ ...
LDA #$AA        change to $BB for ";"
JSR ...
LDA $D01E   ("-" CHAR)
CPX #$26        increase as you like
```

For example, I changed mine just now like this:

```
$C083 C083 DB25:BB N DB2D:46
```

b. Whatever selections you have turned on with the MON command are turned off by the DOS "INT" or "FP" commands. I guess if you want the MONCI modes all the time you could add code to the assembler to set the proper bits inside DOS. The flags are in $AA5E: C=$40, I=$20, O=$10. Store $60 into $AA5E to effect MONCI.

c. I agree with you that it is annoying the way the columns stagger when the line numbers are near 9999. There are several possible solutions. One solution, is to start line numbers at 10000. You can do this by changing the code at $D32B:

```
LDA #990        change to #9990
STA ...
LDA /990        change to /9990
```

A better way is to make a the line numbers always print with five digits. To effect this, change the code at $DE63:

```
LDX #3  change to LDX #4

$C083 C083 DE64:4
```

d. The symbol table does indeed take more space in version 2.0 than it did in previous versions. This is due to the fact that symbols can have values up to 32-bits long. Every symbol has two more bytes in the table now.

e. Right on both counts. Disks with serial numbers 1186 and larger have the corrections you give.

Is there any way of loading a program from the monitor (without going back to Basic) or reload DOS or reboot without losing what is in memory?

Munson Compton, Shreveport, LA

If you entered the monitor via CALL-151 from Basic, or MNTR or MGO-151 from the S-C Macro Assembler, DOS is still alive and will still respond to commands. You can BLOAD or LOAD a program, but of course using LOAD will flip you into either Applesoft, Integer BASIC, or the Macro Assembler depending on file type and what languages are around. If you want to stay in the monitor after the LOAD file has been read into memory, you could temporarily patch the DOS LOAD code which starts at $A413. The book "Beneath Apple DOS" would be helpful here. It looks to me like you could so subvert type A files by patching the JMP ($9D60) at $A44D to RTS (by putting 60 at $A44D). Type I files might be tricked by putting an RTS (60) at $A5AF. I don't know what other ramifications these patches might have. Beware!!!

You can reboot a slave disk without losing the actual text of an assembler source file from memory. However, the pointer which tells the assembler where the program starts will be reset. Before rebooting, record the value stored in $CA and $CB, and after getting back into the assembler restore those two bytes. Of course, if the assembler is in the language card rebooting DOS marks it as not being there. From the monitor you can put it all back by typing:

```
]CALL-151
*C081 C081 E000:20
*INT
:$CA:...(whatever values you recorded earlier)
:LIST   (Voila!)
```

I have the Apple ToolKit and the Big Mac assemblers, and use them primarily to key in source files from articles such as yours. I've figured out how to transpose most of the different labels and opcodes, but would like some enlightenment on the use of the .1, .2, .3 etc. labels that are repeated in the code. I assume this is a capability of your assembler that others don't have.

David Roberson

For help in converting our listings to other assemblers and vice versa, you should refer to my "Directory of Assembler Directives" article in the September 1982 AAL. You are correct in assuming that most other assemblers do not have the kind of local labels as the S-C assemblers, but some do. These numeric labels are one or two digits after a period, and are very convenient for branch points within a sub-routine. They are defined below a normal label, and are only accessible within that area. The local labels are defined internally relative to the preceding normal label, and must be within a 255-byte range after the normal label. Once a new normal label is defined, a whole new set of local labels is available. The use of local labels simplifies programming, because there is no need to think up dozens of unique names like LOOP1, LOP2, LUPA, LUPB, and so on. Local labels also encourage writing good modular code, with only one entry point per module, since the local labels are not accessible outside the routine in which they are defined.

The LISA assembler uses a different type of numeric label, which I call a near-by label. These are redefinable at will, and when they are referenced a pointer must be included to tell the assembler which direction to search for the definition. You can refer to the nearest definition in either a forward or backward direction. I get thoroughly confused trying to read and/or modify programs using these.

Patches Available for Time/Date in Titles.....R. M. Yost

I have implemented a patch to include a Thunderclock (or compatible) time string in the .TItle for version 2.0 of the S-C Macro Assembler. The patch program automatically loads the assembler and my favorite I/O driver, installs the time patch and several others I like, and writes the assembler back on the disk. The new file includes both assembler and driver, with the patches, as well as a loader which allows the whole thing to be executed with a single BRUN.

I will gladly send a listing of the source code to any Assembly Line reader who is interested. Just send a stamped self-addressed envelope to R.M.Yost, 7436 Pointe, Canton, MI 48187.

Write Guard Disk Modification Kit

Mark IV Designs (Mark Hansen) has come up with a neat way to override the write protect switch in a disk drive. Sometimes you want to write on the back side of a disk, in spite of all good breeding. Yet it is a nuisance to have to cut a notch in the other edge of the disk. We finally bought a hole punch, but it is still a nuisance. Other times you want to write protect a disk, but not put one of those little sticky things over the existing notch. What to do?

Instructions for adding an external toggle switch in series or in parallel with the internal sensor are easy to come by, but who wants to drill holes and solder? The Write Guard kit from Mark IV Desings accomplishes all you could wish for without any drilling, cutting, or soldering.

You get a small (1x2x3 inches) box with three-position toggle switch and LED. A short flat cable runs out the back, and you plug that into a socket inside the disk drive (after removing the 74125 from that socket). A piece of velcro attaches the plastic box to either side of your drive. The switch selects normal, always protected, or always unprotected. The LED lights whenever the disk is not protected. One chip on the disk controller card also is replaced with a chip from the kit.

If this kit sounds like something you have been waiting for, you can order one from us at $40.

Review of "Assembly Language for the Applesoft Programmer"
.....reviewed by Bob Sander-Cederlof

Roy E. Myers (author of Microcomputer Graphics) and C.W. Finley, Jr., are the authors of the new book named above, and published by Addison-Wesley. We like it.

Until August of last year we consistently recommended Roger Wagner's "Assembly Lines: the Book" when you asked us which book would best help you learn Apple assembly language. It was especially well-suited to beginners at assembly language who were nevertheless somewhat familiar with the Apple and Applesoft. But it went out of print with the demise of Softalk Publishing, and we can't get them now.

Finley and Myers have not only filled the void, they have improved on our previous favorite. Physically, the book is larger (7x9, paper, 361 + vi pages). It is set in large clear type. And it only costs $16.95 (Wagner's book was $19.95). I especially like the fact that they use the S-C assembler for all of the examples. However, if you don't use our assembler, the book loses no value; all the examples are written so as to be as compatible as possible with other possible assemblers.

Take another look at that title: "Assembly Language for the Applesoft Programmer." There is a double meaning there. This is not only a text for the Applesoft programmer who wants to learn beginning assembly language. It also for the person who wants to USE assembly language along with Applesoft programs. Combining both languages gives the best of both worlds, but doing so involves a lot of work. This book will help.

The book divides into five main sections:

* Introduction

* Fundamentals of 6502 Programming: 6502 architecture, instruction set; addressing; branches, loops, nesting; logical operations and bit manipulation.

* Linkage: fitting a program into the Apple; accessing machine language programs via BLOAD, POKE, USR, ctrl-Y, and "&"; soft switches; using Applesoft ROM subroutines, esp. floating point math; development of a working example.

* Graphics: the Screen, its organization and addressing with text, lo-res, and hi-res; ROM routines for lo- and hi-res graphics; bit-pattern images and animation; bit-masking techniques and complementary drawing; development of a working shoot-em-up video game (GREMLIN).

* Searching and Sorting: &-routine to sort array elements; another to search strings.

There are five useful appendices and an index.

We think enough of this book to add it to our stock. Check our list of books on page 3 for price.

Making Dos-less Disks.....Bob Sander-Cederlof

Last night I re-invented the wheel, and I think I made a pretty good one. I learned a little at the same time.

When you use the DOS "INIT" command, a copy of DOS is written on tracks 0 through 2. If the disk is meant to be a data disk, that wastes three perfectly good tracks. Because of the way DOS checks for the end of track-sector lists and various other things, a standard DOS cannot allow files to be written into track 0. But it is perfectly all right to leave the DOS image off of tracks 1 and 2 and use them for files. Of course it is a good idea to change the image on track 0 so that it will not begin to boot DOS and get lost (when you forget it is DOS-less and try to boot it anyway).

There are some more wasted sectors in track 17, the catalog track. INIT sets up 15 sectors for the catalog, which is enough for 105 files. I have never needed that many, but some of you might have even needed more. Last night I needed only about 30 files, and I needed every sector I could get to store them all. My "wheel" sets up only seven catalog sectors, enough for only 49 files. This frees up eight more sectors for data.

With the help of "Beneath Apple DOS" I examined the code in the DOS File Manager which handles the INIT command ($AE8E-AF07). This routine calls RWTS to initialize 35 empty tracks on a diskette, writes a VTOC in track 17 sector 0 and writes 15 empty catalog sectors on the rest of track 17. Then it scoots back to track 0 and writes the DOS image on the first three tracks.

I used Rak-Ware's DISASM to make a source file out of the INIT code, and then loaded it into the S-C Macro Assembler. Then step-by-step I proceeded to add meaningful labels and comments, and modify the code to do what I wanted.

The File Manager INIT code expects various parameters to have been set up by the DOS command parser, and those will not be set up when my program runs. I decided I would let my program assume that the last disk drive you accessed is the one where you have placed the blank disk you want to initialize.

I also decided to make the volume number always 001. I always do this anyway, and generally consider the volume number to be a nuisance (since I don't have a Corvus which uses the volume numbers for something useful). If you want to be able to choose the volume number, you could add the code for that purpose. Lines 1240-1270 set the volume number into the VTOC image and into the RWTS parameter block (IOB).

Lines 1290-1300 call RWTS to format the blank diskette. Beware! It is entirely too easy to forget to remove your heavily loaded program diskette before running this program! Be absolutely SURE you have the diskette in the drive which you WANT to initialize. After this program runs, the disk will have no remnant of any data which may have been on it before.

Lines 1310-1570 set up a VTOC image. The program assumes that part of the VTOC image at $B3BB is already set up, because you could not run this program without having read at least one VTOC somewhere along the way. The VTOC bitmap is set up first to $FFFF0000 at each sector position, and then the entry for track 0 is cleared. Finally the bits for sector 0 and sectors 9 through 15 of track 17 are cleared. Then lines 1580-1640 call on RWTS to write out the VTOC on track 17, sector 0.

The catalog sectors are chained together with a series of pointers. A pointer in the VTOC points to the first catalog sector, which is almost always track 17 sector 15. A pointer in the first catalog sector points to the second one, and so on. The last catalog sector points at track 0, which is a flag indicating the end of the catalog. (Too bad, because if DOS tested for a final pointer to 0,0 instead of just 0,x we could put the catalog for this data disk all in track 0 and free up even more sectors.)

Lines 1650-1700 clear the catalog buffer, and then lines 1710-1900 insert the forward pointers and call on RWTS to write each sector on the disk.

Finally, lines 1910-2000 write out a bootup program on track 0 sector 0. BOOTER is the code that will be executed if you accidentally try to boot our DOS-less disk.

Lines 2010-2090 finish setting up a call to RWTS, and check for an I/O error. I didn't bother to write any error handler into this program, as you can see by the BRK in line 2090. If you want you can printout the DOS error code at this point, or at least get it in the A-register before the BRK.

The BOOTER program is tricker than it looks. Anyway it tricked me a lot. First notice the .PH and .EP directives in lines 2120 and 2280. These tell the assembler to continue assembling bytes following the preceding code, but to assemble it with the assumption that at execution time it will be origined at $0800. The boot ROM on the disk controller reads track 0 sector 0 into $800-$8FF, so BOOTER has to be set up to run there.

Notice line 2140, which is ".HS 01" The boot ROM reads the first sector into $800-8FF, then checks location $800 to see how many sectors you want the boot ROM to read. About the only disk I have heard of which has anything other than 01 in this byte is the BASICS disk. If you put, for example, 03 in that byte sectors 1 and 2 would be read into $900 and $A00. You can read up to 16 sectors this way, but remember that the sector numbers will not be the same as the ones you use when you write them with RWTS. (RWTS uses a table to convert logical sector numbers into physical sector numbers.)

Line 2150 turns off the disk motor. I forgot the first time, and of course the drive just kept spinning.

Lines 2160-2210 print out the message from lines 2240-2270. My first attempt I called the standard COUT subroutine at $FDED to

print each character, and I lost an hour finding out why I never saw my message. Instead, the drive just kept grinding the head to track 0, over and over and over.... But it worked if I first copied the boot ROM code from $C600 down to $8600, and typed 8600G to boot. I finally figured out that PR#6 sets the output hook to slot 6 and leaves it there. Then the next character that is printed (usually the prompt character for whatever language you are in) through COUT goes to the disk interface and proceeds to boot. My message sent another character to COUT and restarted the boot, ad infinitum. Changing line 2190 to "JSR $FDF0" fixed it all.

After printing the message line 2220 jumps to the initial entry point of the monitor, so you get a "*" prompt. If you previously had DOS in memory, you will probably be able to use 3D0G to get back to BASIC or the assembler or whatever. Otherwise, stick in a disk that DOES have DOS and try booting again.

Line 2300 is just window dressing. It assures that the rest of track 0 sector 0 will have nothing but zeroes in it. No particular value, but I like it that way.

```
                        1000 *SAVE S.DOSLESS INIT
                        1010 *-------------------------------
03D9-                   1020 RWTS        .EQ $03D9
03E3-                   1030 GETIOB      .EQ $03E3
                        1040 *-------------------------------
B3BB-                   1050 VTOC        .EQ $B3BB
B3C1-                   1060 V.VOLUME    .EQ $B3C1
B3EB-                   1070 V.NXTTRK    .EQ $B3EB
B3EC-                   1080 V.DIRECT    .EQ $B3EC
B3F3-                   1090 V.BITMAP    .EQ $B3F3
                        1100 *-------------------------------
B4BB-                   1110 CATALOG.BUFFER .EQ $B4BB
B4BC-                   1120 C.TRACK        .EQ $B4BC
B4BD-                   1130 C.SECTOR       .EQ $B4BD
                        1140 *-------------------------------
B7E8-                   1150 R.PARMS     .EQ $B7E8
B7EB-                   1160 R.VOLUME    .EQ $B7EB
B7EC-                   1170 R.TRACK     .EQ $B7EC
B7ED-                   1180 R.SECTOR    .EQ $B7ED
B7F0-                   1190 R.BUFFER    .EQ $B7F0,B7F1
B7F4-                   1200 R.OPCODE    .EQ $B7F4
                        1210 *-------------------------------
                        1220         .OR $800
                        1230 *-------------------------------
                        1240 DOSLESS.INIT
0800- A9 01             1250         LDA #1        INIT AS VOLUME 001
0802- 8D EB B7          1260         STA R.VOLUME
0805- 8D C1 B3          1270         STA V.VOLUME
                        1280 *-------------------------------
0808- A9 04             1290         LDA #$04      INIT OPCODE FOR RWTS
080A- 20 A3 08          1300         JSR CALL.RWTS.OP.IN.A
                        1310 *---MAKE A GENERIC VTOC----------
080D- A9 11             1320         LDA #$11
080F- 8D EB B3          1330         STA V.NXTTRK
0812- 8D EC B7          1340         STA R.TRACK
0815- A0 01             1350         LDY #1
0817- 8C EC B3          1360         STY V.DIRECT       FORWARD DIRECTION
081A- 88                1370         DEY                Y=0
081B- 8C ED B7          1380         STY R.SECTOR
                        1390 *---PREPARE BITMAP--------------
081E- A0 8C             1400         LDY #4*35
0820- A9 00             1410 .1      LDA #0
0822- 88                1420         DEY
0823- 99 F3 B3          1430         STA V.BITMAP,Y
0826- 88                1440         DEY
0827- 99 F3 B3          1450         STA V.BITMAP,Y
082A- 88                1460         DEY
```

```
082B- A9 FF     1470         LDA #$FF
082D- 99 F3 B3  1480         STA V.BITMAP,Y
0830- 88        1490         DEY
0831- 99 F3 B3  1500         STA V.BITMAP,Y
0834- D0 EA     1510         BNE .1
0836- 8C F3 B3  1520         STY V.BITMAP       CANNOT ALLOCATE TRACK 0
0839- 8C F4 B3  1530         STY V.BITMAP+1
083C- C8        1540         INY                   Y=1, RESERVE F...9
083D- 8C 37 B4  1550         STY 4*17+V.BITMAP          FREE SECTOR 8
0840- A9 FE     1560         LDA #$FE                   RESERVE 0
0842- 8D 38 B4  1570         STA 4*17+V.BITMAP+1        FREE 7...1
                1580 *---WRITE VTOC ON NEW DISK-------
0845- A9 BB     1590         LDA #VTOC
0847- 8D F0 B7  1600         STA R.BUFFER
084A- A9 B3     1610         LDA /VTOC
084C- 8D F1 B7  1620         STA R.BUFFER+1
084F- A9 02     1630         LDA #2             RWTS WRITE OPCODE
0851- 20 A3 08  1640         JSR CALL.RWTS.OP.IN.A
                1650 *---PREPARE CATALOG SECTOR-------
0854- A2 00     1660         LDX #$00
0856- 8A        1670         TXA
0857- 9D BB B4  1680 .2      STA CATALOG.BUFFER,X
085A- E8        1690         INX
085B- D0 FA     1700         BNE .2
                1710 *---WRITE CATALOG CHAIN----------
085D- A9 BB     1720         LDA #CATALOG.BUFFER
085F- 8D F0 B7  1730         STA R.BUFFER
0862- A9 B4     1740         LDA /CATALOG.BUFFER
0864- 8D F1 B7  1750         STA R.BUFFER+1
0867- A9 11     1760         LDA #17        TRACK 17
0869- A0 0F     1770         LDY #15        START IN SECTOR 15
086B- 8D BC B4  1780 .3      STA C.TRACK
086E- 8C ED B7  1790 .4      STY R.SECTOR
0871- 88        1800         DEY
0872- 8C BD B4  1810         STY C.SECTOR
0875- 20 A6 08  1820         JSR CALL.RWTS
0878- AC BD B4  1830         LDY C.SECTOR
087B- C0 09     1840         CPY #9
087D- D0 EF     1850         BNE .4
087F- 8C ED B7  1860         STY R.SECTOR
0882- A0 00     1870         LDY #0
0884- 8C BC B4  1880         STY C.TRACK
0887- 8C BD B4  1890         STY C.SECTOR
088A- 20 A6 08  1900         JSR CALL.RWTS
                1910 *---WRITE BOOT SECTOR------------
088D- A9 B0     1920 .5      LDA #BOOTER
088F- 8D F0 B7  1930         STA R.BUFFER
0892- A9 08     1940         LDA /BOOTER
0894- 8D F1 B7  1950         STA R.BUFFER+1
0897- A9 00     1960         LDA #0
0899- 8D EC B7  1970         STA R.TRACK
089C- 8D ED B7  1980         STA R.SECTOR
089F- 20 A6 08  1990         JSR CALL.RWTS
08A2- 60        2000         RTS
                2010 *-------------------------------
                2020 CALL.RWTS.OP.IN.A
08A3- 8D F4 B7  2030         STA R.OPCODE
                2040 CALL.RWTS
08A6- 20 E3 03  2050         JSR GETIOB
08A9- 20 D9 03  2060         JSR RWTS
08AC- B0 01     2070         BCS .1         ERROR
08AE- 60        2080         RTS
08AF- 00        2090 .1      BRK
                2100 *-------------------------------
                2110 BOOTER
                2120         .PH $800
                2130 BOOTER.PHASE
0800- 01        2140         .HS 01
0801- BD 88 C0  2150         LDA $C088,X  MOTOR OFF
0804- A0 00     2160         LDY #0
0806- B9 14 08  2170 .1      LDA MESSAGE,Y
0809- F0 06     2180         BEQ .2
080B- 20 F0 FD  2190         JSR $FDF0
080E- C8        2200         INY
080F- D0 F5     2210         BNE .1
0811- 4C 59 FF  2220 .2      JMP $FF59
                2230 *-------------------------------
```

```
                    2240 MESSAGE
0814- 8D 8D 87
0817- 87            2250        .HS 8D8D8787
0818- CE CF A0
081B- C4 CF D3
081E- A0 C9 CD
0821- C1 C7 C5
0824- A0 CF CE
0827- A0 D4 C8
082A- C9 D3 A0
082D- C4 C9 D3
0830- CB            2260        .AS -/NO DOS IMAGE ON THIS DISK/
0831- 8D 8D 00      2270        .HS 8D8D00
                    2280        .EP
                    2290 *--------------------------------
08E4-               2300        .BS 256,0
                    2310 *--------------------------------
```

Correction for Symbol Table Source Maker...Bob Sander-Cederlof

I went to great lengths to verify the address of the entry into RENUMBER used by Peter's and Bruce's program, and the day after picking up the printed newsletters Bill discovered that I had used a pre-release copy of Version 2.0. The address in the actual release is different. The correct line 1060 for the version we are sending out is:

```
      1060 RENUMBER .EQ $D658    for the D000 version
   OR 1060 RENUMBER .EQ $1658    for the 1000 version
```

In any case, just be sure the address is the location of the CPX #$06 instruction.

Building Hi-Res Pre-Shift Tables.....Gianluca Pomponi
Pisa, Italy

Given my interest in everything related to graphics, I read eagerly Bob's article "Generating Tables..." in the Dec 94 issue of AAL. I haven't yet had the chance to read the Apple Supplement of Byte (my local newsstand receives it discontinuously); however, I had already heard about the use of preshift tables in animation. I experimented with this technique some time ago, getting excellent results in moving colored shapes against some very complex backgrounds with relatively simple code.

Maybe one of the most challenging steps is typing in the preshift tables. Writing a program to generate the tables is not difficult, and is probably better. The code that follows only takes $68 bytes as a subroutine, using two page zero variables. And it only takes 24 milliseconds to generate the tables, which is many times faster than reading them from a disk.

The Byte article used 14 tables of 256 bytes each. They correspond to left and right portions of each possible 8-bit value shifted any amount from 1 to 7 bits. No columns are kept in memory for shifting 0 bits, as the result is entirely too predictable.

Since, in hi-res graphics, the high bit does not get shifted, you can deal with it separately. Before looking up the preshifted values you can split off the high bit and rejoin it later. The extra code for this is very minor, and it results in a vast memory saving. By doing it this way we get by with 12 tables of 128 bytes each (six pages instead of 14!). Six tables for the left side results and six for the right, for every possible shift of from 1 to 6 bits, for every possible value from $00 to $7F.

I sometimes find it worthwhile to limit the quotient-remainder tables such as Bob generated in the December article to only 256 bytes each (instead of 280), using code like the following to read them when the X-coordinate is larger than 255:

```
LDX XCOORD      low byte of xcoord
LDA QUO+4,X
CLC
ADC #$24
STA XBYTE
LDA REM+4,X
STA XBIT
```

Here now is my program to generate the preshift tables, as modified by Bob. LInes 1080-1210 allocate space for the 12 tables, each 128 bytes long. I put them at $0900 for this example, but of course you can put them wherever you wish.

Lines 1230-1310 are a macro definition. The macro is called out six times in the main loop, once for each shift of a value. For the benefit of those without a macro assembler, I have

shown the expansion in the listing of lines 1430-1480. Some of the code in the macro could have been handled by a subroutine, but it would save a negligible amount of space at a cost of an non-negligible amount of time.

The shifting algorithm is familiar to those of you who have been fiddling with hi-res for a while. Remember that the picture bits are stored backwards in each byte, so that shifting the picture on the screen right one bit requires shifting the bits in memory left within each byte, stepping over bit 7, and from byte to byte in a left-to-right direction.

The little program called TIME, lines 1530-1660, calls the BUILD program 1000 times. I ran it and clocked it at a little less than 24 seconds, which means building once took less than 24 milliseconds. The tables would take up six disk sectors if they were stored part of the program on disk. The disk spins at 300 rpm, or 200 milliseconds per revolution. The absolute minimum time to read six sectors would be 67.5 milliseconds, but in actual practice it takes closer to a half second. It depends on whether it is part of a larger file or stored as a separate file, the latter taking longer. Since the program only needs to be executed once, even the memory it occupies it available to the program for other purposes.

```
                1000 *SAVE S.BUILD.PRESHIFT.TABLES
                1010 *--------------------------------
                1020 *   WRITTEN BY G. L. POMPONI, PISA, ITALY
                1030 *   MODIFIED BY BOB SANDER-CEDERLOF
                1040 *--------------------------------
00-             1050 L.BYTE .EQ 0
01-             1060 R.BYTE .EQ 1
                1070 *--------------------------------
                1080         .OR $900
0900-           1090 SHIFT.1 .BS 128
0980-           1100 SHIFT.2 .BS 128
0A00-           1110 SHIFT.3 .BS 128
0A80-           1120 SHIFT.4 .BS 128
0B00-           1130 SHIFT.5 .BS 128
0B80-           1140 SHIFT.6 .BS 128
                1150 *--------------------------------
0C00-           1160 REMND.1 .BS 128
0C80-           1170 REMND.2 .BS 128
0D00-           1180 REMND.3 .BS 128
0D80-           1190 REMND.4 .BS 128
0E00-           1200 REMND.5 .BS 128
0E80-           1210 REMND.6 .BS 128
                1220 *--------------------------------
                1230         .MA SHIFT
                1240         ASL L.BYTE
                1250         ROL R.BYTE
                1260         LDA L.BYTE
                1270         LSR
                1280         STA SHIFT.]1,X
                1290         LDA R.BYTE
                1300         STA REMND.]1,X
                1310         .EM
                1320 *--------------------------------
                1330         .OR $800
                1340 *--------------------------------
                1350 BUILD.PRESHIFT.TABLES
0800- A2 00     1360         LDX #0        FOR X = 0 TO $7F
                1370 *--------------------------------
0802- 86 00     1380 .1      STX L.BYTE
0804- A9 00     1390         LDA #0
0806- 85 01     1400         STA R.BYTE
0808- 06 00     1410         ASL L.BYTE
                1420 *--------------------------------
```

```
080A-             1430         >SHIFT 1
080A- 06 00       0000>         ASL L.BYTE
080C- 26 01       0000>         ROL R.BYTE
080E- A5 00       0000>         LDA L.BYTE
0810- 4A          0000>         LSR
0811- 9D 00 09    0000>         STA SHIFT.1,X
0814- A5 01       0000>         LDA R.BYTE
0816- 9D 00 0C    0000>         STA REMND.1,X
0819-             1440         >SHIFT 2
0819- 06 00       0000>         ASL L.BYTE
081B- 26 01       0000>         ROL R.BYTE
081D- A5 00       0000>         LDA L.BYTE
081F- 4A          0000>         LSR
0820- 9D 80 09    0000>         STA SHIFT.2,X
0823- A5 01       0000>         LDA R.BYTE
0825- 9D 80 0C    0000>         STA REMND.2,X
0828-             1450         >SHIFT 3
0828- 06 00       0000>         ASL L.BYTE
082A- 26 01       0000>         ROL R.BYTE
082C- A5 00       0000>         LDA L.BYTE
082E- 4A          0000>         LSR
082F- 9D 00 0A    0000>         STA SHIFT.3,X
0832- A5 01       0000>         LDA R.BYTE
0834- 9D 00 0D    0000>         STA REMND.3,X
0837-             1460         >SHIFT 4
0837- 06 00       0000>         ASL L.BYTE
0839- 26 01       0000>         ROL R.BYTE
083B- A5 00       0000>         LDA L.BYTE
083D- 4A          0000>         LSR
083E- 9D 80 0A    0000>         STA SHIFT.4,X
0841- A5 01       0000>         LDA R.BYTE
0843- 9D 80 0D    0000>         STA REMND.4,X
0846-             1470         >SHIFT 5
0846- 06 00       0000>         ASL L.BYTE
0848- 26 01       0000>         ROL R.BYTE
084A- A5 00       0000>         LDA L.BYTE
084C- 4A          0000>         LSR
084D- 9D 00 0B    0000>         STA SHIFT.5,X
0850- A5 01       0000>         LDA R.BYTE
0852- 9D 00 0E    0000>         STA REMND.5,X
0855-             1480         >SHIFT 6
0855- 06 00       0000>         ASL L.BYTE
0857- 26 01       0000>         ROL R.BYTE
0859- A5 00       0000>         LDA L.BYTE
085B- 4A          0000>         LSR
085C- 9D 80 0B    0000>         STA SHIFT.6,X
085F- A5 01       0000>         LDA R.BYTE
0861- 9D 80 0E    0000>         STA REMND.6,X
                  1490 *-------------------------------
0864- E8          1500          INX             NEXT X
0865- 10 9B       1510          BPL .1          (...UNTIL $80)
0867- 60          1520          RTS
                  1530 *-------------------------------
                  1540 *   BUILDS 1000 TIMES IN LESS THAN 24 SECONDS,
                  1550 *      SO LESS THAN 24 MILLISECONDS TO BUILD ONCE
                  1560 *-------------------------------
0868- A9 04       1570 TIME     LDA #4          4*250 = 1000
086A- 8D 00 05    1580          STA $500
086D- A0 FA       1590 .1       LDY #250
086F- 20 00 08    1600 .2       JSR BUILD.PRESHIFT.TABLES
0872- 88          1610          DEY
0873- D0 FA       1620          BNE .2
0875- CE 00 05    1630          DEC $500
0878- D0 F3       1640          BNE .1
087A- 60          1650          RTS
                  1660 *-------------------------------
```

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $12 postage for other countries. Back issues are available for $1.80 each (other countries add $1 per back issue for postage).